CSE-681 Software Modelling and Analysis

Instructor- Dr. Jim Fawcett

PROJECT #1

Build Server

Operational Concept Document

Harika Bandaru(SUID# 47266909)

09/13/2017

Contents

PROJECT #1 0
PROJECT #1 0
1. Executive Summary 3
2. Introduction 4
2.1. Objective and Key idea 4
2.2. Application Obligations 4
2.3. Organizing Principles 5
3. Uses & Users 5
3.1. Uses 5
3.2. Users 5
3.2.1. Users of finished Version 5
Developer's: 5
Quality Assurance Team 6
Manager needs for progress information 6
3.2.2. Instructional Users 6
Developer 6
Teaching Assistant/Instructor 6
3.3. Extending to Project #3 and Project#4 6
4. Application Activities 6
4.1. Client Process 8
4.2. Logs 8
4.3. Test Request 8
4.4. The Build Server 8
4.4.1. Parse Test Request 10
4.4.2. Create directory 10
4.4.3. Build Test Library 10
4.5. Test Harness 10
4.6. Display results 10
5. Partitions 11
5.1. Client Process (UI) 12
5.2. XML Manager 12

Parser: 13
5.3. Log Manager: 13
5.4. Directory Manager 14
5.5. Build Server 14
5.6. Test Harness 14
5.7. Repository 14
5.8. Display 15
6. Critical Issues 15
6.1. Ease of Use 15
6.2. Design Issues 15
6.2.1. Spawning a Process 15
6.2.2 Cache at Test Harness 16
6. User Interface 16
8. APPENDIX 17
8.1 Prototype: Build a project programmatically 17
8.2. Things Learned 18
9.References 20

Figures

Figure 1 : Activity diagram of Build Server (Project #4) 7
Figure 2 : Activity diagram of Build Server 9
Figure 3: Package diagram for Build Server 11
Figure 4 : User Interface 16

1. Executive Summary

The purpose of this project is to implement the Build Server which is intended to build test libraries based on the test request generated by the client process and the code sent by the repository. On completion of build process, if successful, the build server sends the test libraries to the Test Harness, sends build logs to the Repository.

The Build Server initially accepts the test request consists of a message, perhaps in XML format, that lists one or more tests to execute. Build Server runs a parser to parse the XML file to know how each test is configured. It further requests repository to send test drivers, source code on which tests needed to be executed, and creates a Test Library with the information provided by the Repository.

The Build Server should provide the following key capabilities:

- Accept the test request from client process.
- Parse the test request to know the test configuration details, create a temporary directory and pick the tool chain, to make Build Server language independent. So, that the Build Server have the capability to build C#, CPP, Java projects.
- Send request to repository and store the test drivers and source code files from repository in temporary directory created.
- Build Server tries to build test library on the source test code and drivers mentioned in the test request.
- On build success, the process should return the test library to test harness, test harness will execute the test library.
- Generate build logs, provide a log storage facility in repository and send a notice to the client process.
- Remove the unique temporary directory created during the process of building a Test Library.
- Build Logs helps developer to analyze the Build Status more efficiently.

Below are the some of the critical issues associated with the project and solutions are discussed in detail in further sections

- Number of process that can spawned at startup.
- Implementation of cache at Build Server, leads to lack of synchronization between files present in cache and repository.
- Implementation of cache to store Test Libraries at Test Harness can lead to lack of synchronization between newly created Test Library with modified source code and the older version present in cache of Test Library.
- Ease of Use: An application should provide easy access to the users and should be user friendly.
- Demonstration of an application meeting all the requirements and in an optimal way possible.

2. Introduction

Over the last few years we have witnessed an exponential growth in spawning ideas into software be it an Application or a System software. In order to successfully implement these software systems, we need to partition code into relatively small parts and thoroughly test each of the part before integrating into the software baseline to detect the issues before proceeding to the later phases of development. This is an essential step as cost of finding and fixing the issues increases exponentially from development phase to deployment phase. This brought the importance of periodic builds into focus. It started with weekly build. Then it grew tighter when nightly began, and still tighter with hourly. As the benefits of frequent builds became more obvious, organizations wanted more builds. And now we have continuous integration builds. The Build Servers or Continuous integration servers serving as heart monitors in the process of developing big software systems by implementing Continuous Integration builds.

2.1. Objective and Key idea

The objective of the project is to implement the Build Server to build a test library, provides an executable library to Test Harness. Test Harness runs sequence of tests on code present in the Repository project#2. extending to project #3 and project#4 providing GUI to the Client . Establishing communication between the key modules in the application (Repository, Test Harness, Build Server, Client Process) with the help of Windows Communication Foundation. In project#4 a mother Build Server sits on the top of Build Server processes extending the capability of running multiple Build Servers to handle multiple Test Requests at a time.

2.2. Application Obligations

The main responsibility of this application is to provide:

- A repository capable of storing Log files, source code, test drivers and test stubs, it also fetches information regarding log files, on request, by the client process.
- In the project#2 user directly interacts with Build Server. Project #3, we provide GUI for user to efficiently interact with Build Server along with key modules.
- Parse the Test Request, which is an XML file to see how each test is configured.
- Establish an environment based on the tool chain present in the test request to support building of Test Libraries independent of language (Java, C#, CPP etc.)
- Create a temporary directory to save the files send by the repository on Build Server Request.
- Spawn a process from “Process Pool” that handles test library creation process when client sends multiple test requests. The process will have all the functionalities of Build Server implemented in project#2
- Invoke Test Harness with a test library created by the spawned process.

- Accept log files from Test Harness and the Build Server and store them in repository. These are essential records required during analysis of an issue.
- Notify the client process regarding the status of the Build Sever and Test Harness Server.

2.3. Organizing Principles

This system will be implemented in C# using facilities of the .Net framework, Windows Communication Framework(WCF) for establishing communication between Repository, Build Server, Test Harness and the Client process, Windows Presentation Foundation(WPF) to implement GUI for client process, Visual Studio 2017.The key modules are Build Server, Repository, Test Harness and Client Process. The functionalities of each module are implemented through individual packages, which will be explained in detail in the later sections.

3. Uses & Users

3.1. Uses

- **User Interface** – Helps user to build a test library by just creating or selecting a Test Request from the GUI and proves to be an easy way to access repository.
- **User Friendly** – It does not require the user to know the language before using the tool.
- **Build Log and Test Log** – Build Server and Test Harness removes the overhead of manually generating the reports and comparing them with desired output by providing build log and test log respectively. These reports are stored in a well formatted structure which can be directly used for analyzing purposes.
- **Quality** – Automatic process of building will remove the chances of human error which can occur while building the code in their local environment.

3.2. Users

There are two kinds of users we need to address:

Users of finished version of the final project, and instructional users for each of the project through project #2 to project #4.

3.2.1. Users of finished Version

Users of the finished product are defined by its use in a software development organization:

Developer's:

Developers of big software system use build server more frequently than any other group to have a faster release and better-quality software. As the problems for the build break are identified earlier and is easy to fix the broken builds at initial stage. As the Build Server can replicate your target environment and there is less chance of something working on developers' desktops and breaking in production.

Quality Assurance Team

QA team uses Build Server to create builds to execute different types of testing like continuous integration testing, regression testing etc. to make code standard analysis and to look for structural defects.

Manager needs for progress information

Manager uses Build Server to know whether the systems build by developers satisfies the code quality from the logs files generated by the Build Server and Test Harness Server. These log files are stored in the Mock Repository.

3.2.2. Instructional Users

Instructional uses have different actors, the Developing student and Teaching Assistant/Instructor, with different needs. Each student needs to thoroughly test each part of the developing project and, at the end, demonstrate each requirement to the instructor. The instructor will run each project and look at its code implementation for evaluation.

Developer

The developing Student is intended to understand all the requirements and should demonstrate all the requirements through project#2 to project#4. Analyze all the critical issues that project can face while implementing, should provide feasible solutions to all the critical issues. The Developers can use this document to extend the Build Server to communicate with Repository and with Test Harness to extend the functionalities provided by each of them.

Teaching Assistant/Instructor

The TA will use the application to check whether the Student Developer have successfully demonstrated all the requirements addressed in the Operational Concept Document or not. They also verify the critical issues identified by the Student Developer and the solutions to those issues are best addressed or not. TA’s will also verify the feasibility of application for future projects.

3.3. Extending to Project #3 and Project#4

In the project#3 we will provide a GUI using Windows Presentation Foundation(WPF) . Client can easily build the test request using the GUI. A communication mock is implemented using Windows Communication Framework(WCF). In project#4 will be using WCF to establish a smooth communication between Client process, Build Server, Repository, Test Harness. We also extend the functionality of Build Server to spawn a process that acts like a Build Server implemented in project#2, to manage multiple incoming Test Requests.

4. Application Activities

The motto of this application is to implement the Build Server which is part of Software Collaboration Federation. We also use different modules like Repository, Test Harness and the Client process. The overview of flow of activities for this project is expressed in the below activity diagram.

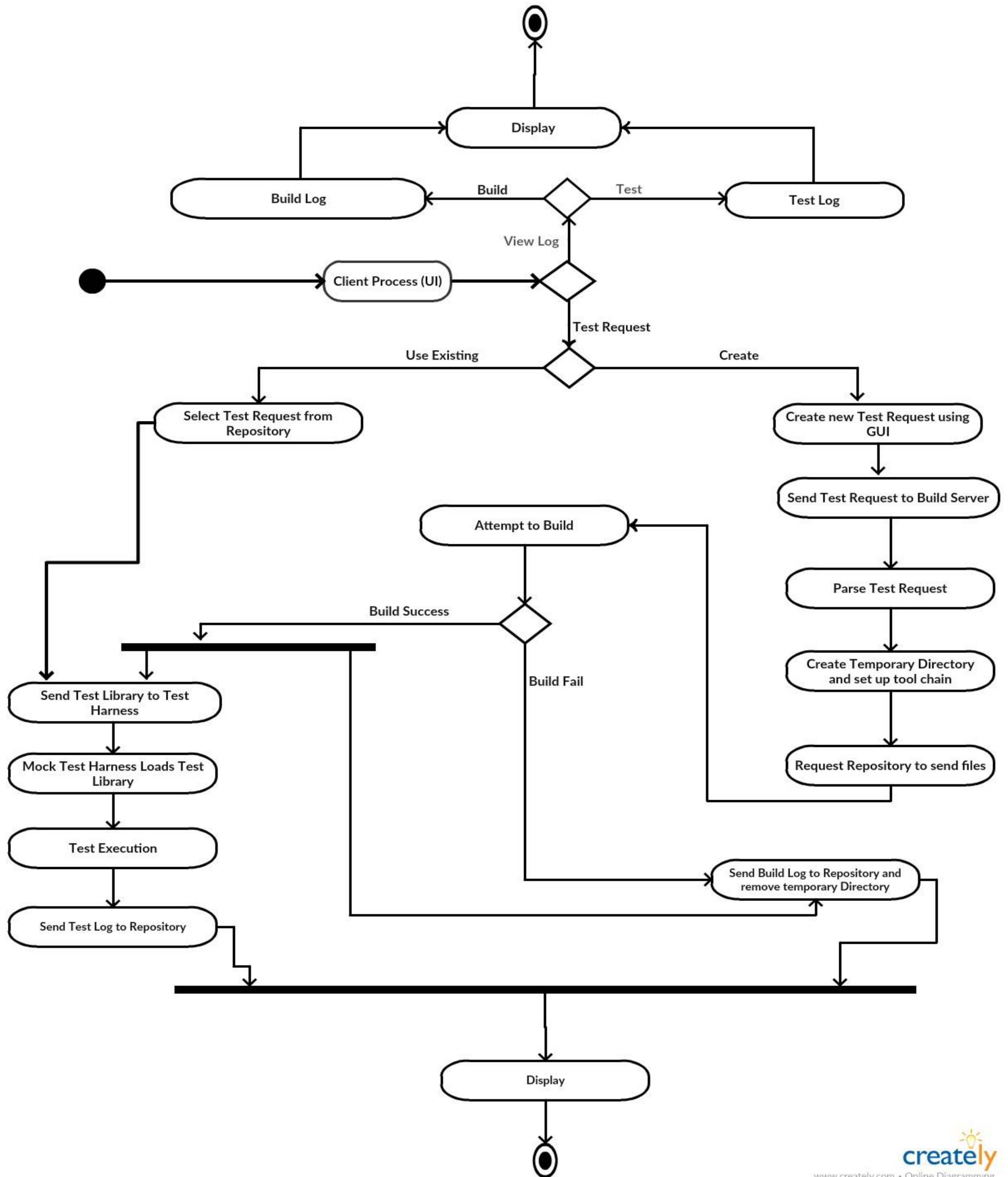


Figure 1 : Activity diagram of Build Server (project #4)

The main activities involved in the application are described below in the sequence.

4.1. Client Process

The client process will provide User Interface to select whether the client wants to view the logs (build and test) or to generate a test request. Depending on the choice of the user the Client Process will start the other processes. It provides an efficient GUI which makes creation of Test Request simple. Client process also shows the notifications send by the Build Sever and Test Harness.

4.2. Logs

The Logs are stored in the Repository. These Logs helps to examine the cause for failure of Build or the number of tests that got failed. From this information, we can analyze the causes for build failure and the test failure. It provides information about the latest successful run. It makes the process of debugging easier and eliminates running the debugger program to know the causes of failures.

4.3. Test Request

The client can make his own test request from GUI provided by Client Process. It also provides the capability of selecting existing test requests present in Repository to user. User can also view these Test Requests.

<u>Basic Test Request</u>:

```
▼<testRequest>
   <author>Jim Fawcett</author>
   <dateTime>9/7/2017 9:49:14 PM</dateTime>
  ▼<test>
     <testDriver>td1.cs</testDriver>
     <tested>tf1.cs</tested>
     <tested>tf2.cs</tested>
     <tested>tf3.cs</tested>
   </test>
 </testRequest>
```

4.4. The Build Server

In project #2 the main motto is to implement the Build Server. The Build Server is mainly used to prevent Integration problems at an early stage of the development of Big Software Systems which aims to provide a quality software. The Build Server is an automated tool to build Test Libraries.

The Build Server may have heavy workloads just before the customer demos and releases. We want to make the throughput for building code as high as is reasonably possible. To do that the build server will use a "Process Pool". A limited set of process are spawned at startup. The number of process that need to be spawned is specified by the user in the configuration file. The parent Build Server provides a set of Test Requests, and each pooled process retrieves a request from Test Request queue, processes it and sends the build log and, if successful sends libraries to test harness, then retrieves another Test Request.

The selection of process from "Process Pool" and assigning a "Test Request" from Test Request queue will be implemented in project#4. Every process has the same functionalities of the Build Server implemented in project#2.

The overview of flow of activities for the Build Server in project#2 is expressed in below activity diagram.

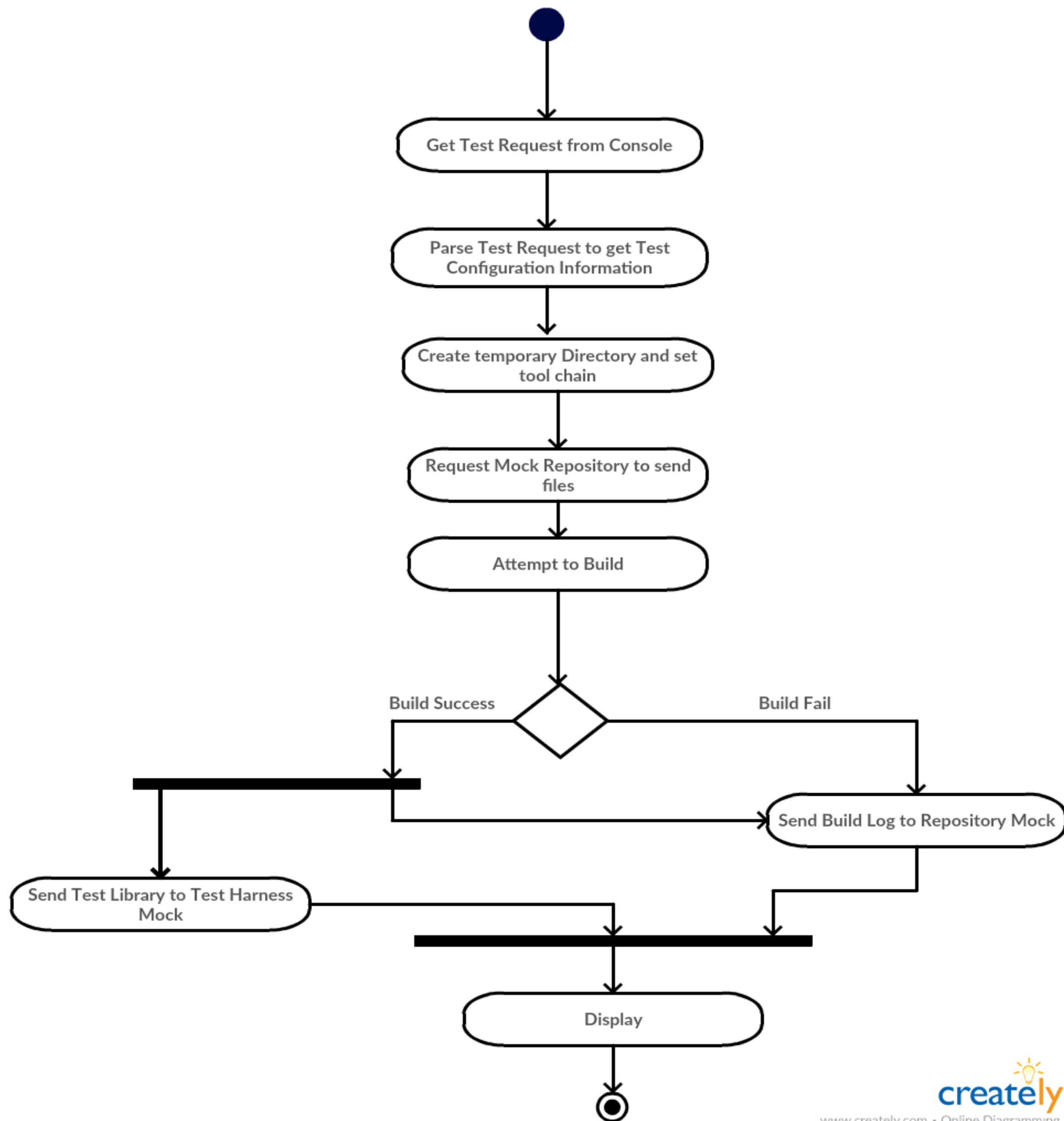


Figure 2: Activity diagram of Build Server

4.4.1. Parse Test Request

Test Request is an XML file to get the contents of XML file we need parse. C# with .Net framework provides many namespaces and classes to perform the above-mentioned task. In this Project will be using XmlDocument class present in System.Xml namespace.

Example:

XmlDocument xmlDoc = new XmlDocument (); //Create an XML document object

xmlDoc.Load("testRequest.xml"); //Load the XML document from the specified file

XmlNodeList tested = xmlDoc.getElementsByTagName("tested"); //Contains list of tested files

4.4.2. Create directory

A temporary directory is created with a unique name to store the files, after parsing the Test Request, Build Server requests the Repository to send files. The files sent by the Repository are stored in this directory and are used by Build Server.

4.4.3. Build Test Library

The main functionality of the Build Server is to build the test library. On receiving the test request the Build Server spawns a process(project#4) to start the activity on build success it should notify the client process, commands the test harness to run and store the build log to the repository. If the build fails the process will notify to client, send build log to repository which helps to user to analyze the causes for build failure.

In project #2, Build Server parses the Test Request using XML Parser and creates a Test Library. On successful, creation of Test Library it commands Test Harness. It notifies the client by sending the status of the build and sending Repository to store Build Log.

4.5. Test Harness

On successful creation of test library test harness will load the libraries, execute the tests specified in the library and send the Test Log to the Repository and notify the Client with number of tests passed and failed.

4.6. Display results

All the requirements shall be demonstrated successfully through a series of discrete tests with display to the console. Notifications from the Build Server and the Test Harness are displayed to the user through console.

5. Partitions

Based on the functionality, all the activities or tasks are split into different packages which have independent roles to play. The following are the packages that belong to the Build Server a server of Software Collaboration Federation with Repository, Test Harness acting as mock and a Client Process.

Below is the package diagram for Build Server.

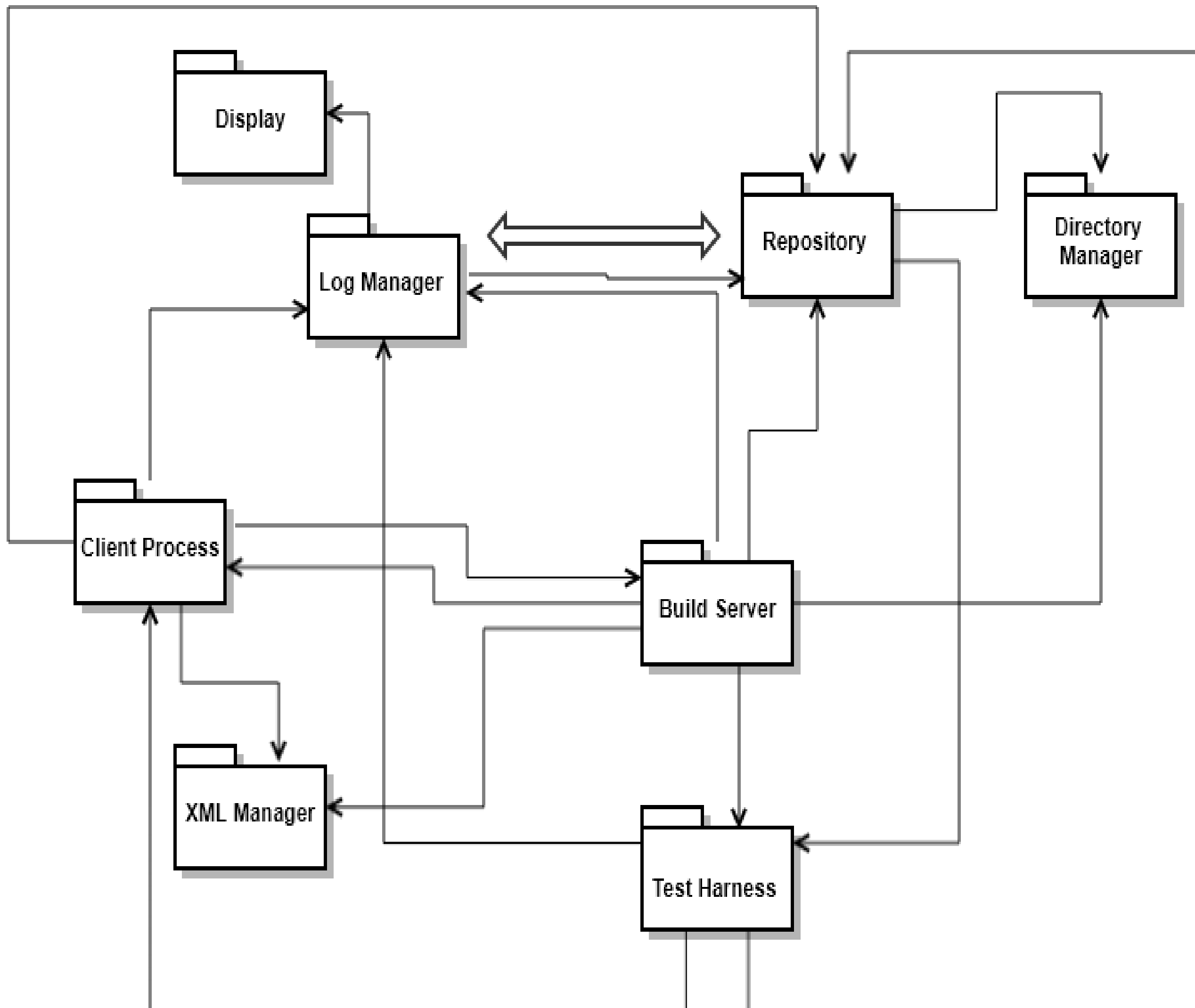


Figure :3 Package diagram for Build Server

5.1. Client Process (UI)

Client Process acts as the entry point to the Build Server application. It provides Graphical User Interface(GUI) to the client to interact with the application. It provides a high-level view of the total application. It can access Log Manager package to view logs, can request Repository for existing test requests, send created test requests to Build Server.

Client Process initiates below sequence of actions:

- User can view build or test logs.
- Create a Test Request using XML Manager package.
- Send the Test Request to Build Server.

5.2. XML Manager

This package provides two key functionalities:

- To create/override the Test Request which is an XML file.
- Parsing of Test Request to know the Test Configuration information.

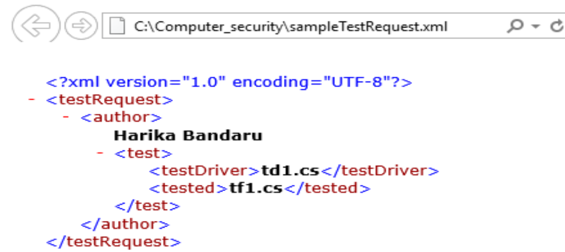
A GUI is provided to create a Test Request it takes inputs and generate an XML file based on the user input. It also displays the Test Requests present in the repository, allows user to view the test request file to see how each test is configured.

Code Snippet for creation of Test Request using XmlDocument:

```
XmlDocument doc = new XmlDocument (); //Predefined class to create XML file


XmlElement element0 = doc.CreateElement(string.Empty, "testRequest", string.Empty);
doc.AppendChild(element0);
XmlElement element1 = doc.CreateElement(string.Empty, "author", string.Empty);
XmlText text1 = doc.CreateTextNode("Harika Bandaru");//The value to be taken from GUI
provided
element1.AppendChild(text1);
element0.AppendChild(element1);
XmlElement element2 = doc.CreateElement(string.Empty, "test", string.Empty);
element1.AppendChild(element2);
XmlElement element3 = doc.CreateElement(string.Empty, "testDriver", string.Empty);
XmlText text3 = doc.CreateTextNode("td1.cs");//User have to choose the value
element3.AppendChild(text3);
element2.AppendChild(element3);
XmlElement element4 = doc.CreateElement(string.Empty, "tested", string.Empty);
XmlText text2 = doc.CreateTextNode("tf1.cs");//User have to choose the value
element4.AppendChild(text2);
element2.AppendChild(element4);
doc.Save("C:\\Computer_security\\sampleTestRequest.xml");
```

Output of Above Code Snippet:



Parser:

C# with .Net framework provides a large collection classes in the System.Xml namespace like XmlTestReader, XmlDocument etc.

For example:

Code Snippet of Test Request in XML format:

```
▼<testRequest>
   <author>Jim Fawcett</author>
   <dateTime>9/7/2017 9:49:14 PM</dateTime>
  ▼<test>
     <testDriver>td1.cs</testDriver>
     <tested>tf1.cs</tested>
     <tested>tf2.cs</tested>
     <tested>tf3.cs</tested>
   </test>
 </testRequest>
```

Parsing of above mentioned Test Request:

Example:

XmlDocument xmlDoc = new XmlDocument (); //Create an XML document object

xmlDoc.Load(“testRequest.xml”); //Load the XML document from the specified file

xmlDocNodeList author = xmlDoc.getElementsByTagName(“author”); //information about the author.

xmlDocNodeList testDriver = xmlDoc.getElementsByTagName(“testDriver”); //Getting test driver

XmlNodeList tested = xmlDoc.getElementsByTagName(“tested”); //Contains list of tested files

5.3. Log Manager:

The main functionalities of this package are:

- Provide logging information to Repository from Build Server and Test Harness.

- Display the logs based on user choice.
- Display package is called by the Log Manager package to display the logs whenever user requests for log information.

Logging makes debugging process much easier and smoother.

5.4. Directory Manager

The functionality of this package is to create a unique directory to store test code and test drivers send by the repository, these are the necessary files required by the Build Server to create a Test Library. This directory is removed once the process commands Test Harness with the test library on successful creation of Test Library. And if build fails once the Log is sent to Log Manager directory is removed.

- Interacts with Build Server to supply required files to build Test Library
- Interacts with Repository to save files send by Repository on request from Build Server.

5.5. Build Server

This is the main package whose functionalities adds up from project #2 through project#4. Below are key functionalities of the Build Server:

- Accept test request from the Client Process (in project #4) and from console (project#2)
- Interacts with XML Manager to parse the Test Request, an XML file.
- Interacts with Directory Manager Package to create a temporary directory.
- Interacts with Repository Package to fetch test code files to temporary directory created by Directory Manager package.
- In Project #4 spawns a process which mimics the functionality of Build Server in Project # 2 and maintains a Test Request queue.
- Check the status of the process to allocate a Test Request from Test Request queue.
- Creates Test Library which then send to Test Harness.
- Interacts with Log Manager to send Log reports and Client process to notify the status.

5.6. Test Harness

The functionality of this module is to test the library generated by the Build Server by setting the tool chain from the information provided in the test request. If there are large number of test libraries generated by the Build Server it should maintain a test library queue.

- Interacts with Build Server to get Test Library.
- Interacts with Log Manager to send Test log to Repository.

5.7. Repository

This package provides one of the main functionalities to the Build Server.

- To start with, it facilitates **Client Process** which provides user interface to fetch existing Test Requests.

- Will facilitate **Log Manager** package to save logs to the Repository and to retrieve logs from the Repository.
- Interacts with Build Server by providing the requested files.

5.8. Display

This package is used to demonstrate whether all requirements specified are met or not. It displays the result of various packages. It is mainly invoked by Log Manager to display the log information which are essential to check the breakage in code and the test execution process from build log and test log respectively.

6. Critical Issues

6.1. Ease of Use

This one of the major issue for any application or project that is to be developed. The main goal is to build an application that is clever to perform functions in an optimal way possible but from the client point of view it should be very easy to use. The functionalities provided by the application should be clear to the user to understand and make use of them.

Build Server should be **User Friendly** not user specific. Even a novice developer should use Build Server without prior knowledge of "how it works?" This can be achieved by providing a clear and easily understandable User Interface. By proper use of buttons, those allows users to navigate from one functionality to another. Displaying text clearly and in proper way rather than using acronyms etc.

6.2. Design Issues

6.2.1. Spawning a Process

When there is a heavy load on build server especially just before customer demos and releases. If the Mother Build Server goes on spawning the processes to handle incoming test requests. The purpose of spawning a process will go vain as process keep on context switching between themselves rather than carrying out the functionality of the Build Server.

Solution: Instead of continuously spawning the process as soon as the Test Request is parsed, it should limit the number of process spawned at a startup depending on Request queue size and configuration file. Should limit the number of process spawned to the twice the maximum number of process a system can support.

This value can be supplied through the application configuration file.

This issue can occur both at Build Server and at the Test Harness.

6.2.2 Cache at Test Harness

If we maintain a cache at Test Harness to store the Test Library if the Test Request contains an updated test code, instead of creating a new Test Library, Build Server simply commands Test Harness to execute existing Test Library. In this case we are missing those tests executed on updated code.

Solution: The best way to avoid this conflict is to maintain a **versioning system** at Repository. Every time the developer needs to Check-In the updated code a new version will be allocated. As versions are immutable there is no conflict between the Test Library in cache and test configuration present in the Test Request so the Build Server builds a new Test Library.

This issue occurs at **Build Server** also, if it tries to maintain a cache for files send by the Repository.

6. User Interface

Overview of User Interface that will be implemented in project#3.

Below are the sample screens how an application looks to the end user.

- Start Window is the Home page of build server. A user can either View Logs, or submit Test Request to the Build Server.
- If the user selects the View Logs he can decide whether he wants to view build logs, test logs of the latest runs depending on the time interval provided by the user.
- User can create a test request or select an existing test request
- User Interface is provided to create a test request which makes user task much easier.

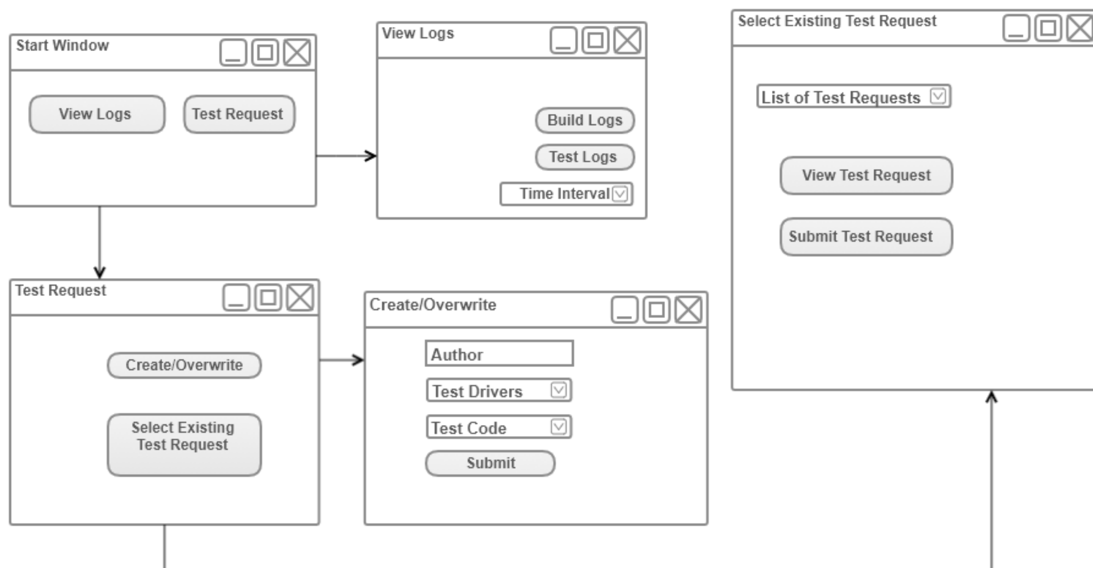


Figure 4 : User Interface

8. APPENDIX

8.1 Prototype: Build a project programmatically

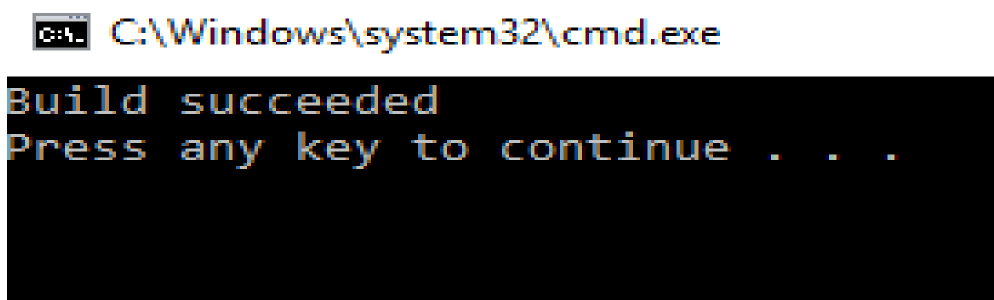
This is to implement a program that attempts to build the project from the specified path. It should also show the build log by writing to a file or on a console with Build Status.

Used the Engine class which is present in the reference “Microsoft.Build.BuildEngine”. Used FileLogger class of Miscrosoft.Build.Framework to log the build log information to the file.

Below is the prototype code snippet for building a project through program:

```
28      {
29
30          Engine engine = new Engine();
31          String path = ConfigurationManager.AppSettings.Get("binPath");
32          FileLogger logger = new FileLogger();
33          logger.Parameters = @"logfile=../../buildlog.txt";
34          engine.RegisterLogger(logger);
35          bool success = engine.BuildProjectFile("../../HelloWorld/HelloWorld.sln");
36          engine.UnregisterAllLoggers();
37          if (success)
38              Console.WriteLine("Build succeeded");
39          else
40              Console.WriteLine("Build Failed");
41
42      }
```

Output for the above code snippet:



AppConfig File:

```
App.config   BuildProgramatically.cs
1  <?xml version="1.0" encoding="utf-8" ?>
2  <configuration>
3      <startup>
4          <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
5      </startup>
6  <appSettings>
7    <add key ="binPath" value ="C:\Windows\Microsoft.NET\Framework\v4.0.30319"/>
8  </appSettings>
9  </configuration>
```

BuildLog.txt file generated:

```
buildlog - Notepad
File  Edit  Format  View  Help
MSBUILD : warning MSB4056: The MSBuild engine must be called on a single-threaded-apartment. Current threading model is "MT.
Build started 9/13/2017 12:13:43 PM.
__________________________________________________
Project "C:\Users\Harika\source\repos\BuildPrototypev2.0\BuildPrototypev2.0\HelloWorld\HelloWorld.sln" (default targets):

Target ValidateSolutionConfiguration:
    Building solution configuration "Debug|Any CPU".
Target Build:
    __________________________________________________
    Project "C:\Users\Harika\source\repos\BuildPrototypev2.0\BuildPrototypev2.0\HelloWorld\HelloWorld.sln" is building "C:\

    Project file contains ToolsVersion="4.0". This toolset is unknown or missing. You may be able to resolve this by instal
    Target CoreCompile:
      Skipping target "CoreCompile" because all output files are up-to-date with respect to the input files.
    Target _CopyAppConfigFile:
      Skipping target "_CopyAppConfigFile" because all output files are up-to-date with respect to the input files.
    Target CopyFilesToOutputDirectory:
        HelloWorld -> C:\Users\Harika\source\repos\BuildPrototypev2.0\BuildPrototypev2.0\HelloWorld\HelloWorld\bin\Debug\He

Build succeeded.

MSBUILD : warning MSB4056: The MSBuild engine must be called on a single-threaded-apartment. Current threading model is "MT.
    1 Warning(s)
    0 Error(s)
```

8.2. Things Learned

The major learning while implementing this prototype was about how **Microsoft.Build.BuildEngine** and **Microsoft.Build.BuildFramework** namespaces are used.

- Used an Engine class present in Microsoft.Build.BuildEngine namespace to build the project file specified.
- Used FileLogger to log the build status along with warnings and errors to a "**buildlog.txt"** file.
- Learned how to get values from appConfg file, as the path to MSBuild.exe need to be specified to Engine parameter "BinPath"
- Learned creation of XML using XmlDocument class as the program need to be build is an XML creation program.

9.References

1) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project1HelpF2017/
2) https://msdn.microsoft.com/en-us/library/microsoft.build.buildengine.filelogger.aspx
3) https://stackoverflow.com/